

FEATURE ARTICLE

Jim Sibigroth

Fuzzy Logic for Embedded Microcontrollers

Fuzzy logic doesn't necessarily need lots of horsepower. Many embedded applications that use more traditional control schemes can benefit from the use of fuzzy logic. Jim looks at how to keep things simple and speedy.

after describing basic fuzzy-logic concepts, this article explains how to implement fuzzy-inference algorithms in a general-purpose embedded controller. The examples, written in assembly language, are for an MC68HC11, but the algorithms could be adapted for any general-purpose microcontroller. Code size is surprisingly small and execution time is fast enough to make fuzzy logic practical even in small embedded applications.

Perhaps because of its strange sounding name, fuzzy logic is still having trouble getting accepted as a serious engineering tool in the United States. In Japan and Europe, the story is quite different. The Japanese culture seems to respect ambiguity, so it is considered an honor to have a product which includes fuzzy logic. Japanese consumers understand fuzzy logic as intelligence similar to that used in human decisions.

In the US, engineers typically take the position that any control methodology without precise mathematical models is unworthy of serious consideration. In light of all the fuzzy success stories, this position is getting hard to defend.

I think the European attitude is more appropriate. It recognizes fuzzy

logic as a helpful tool and uses it. They regard the difficulties of the nomenclature as a separate problem. Since the term "fuzzy" has negative connotations, they simply don't advertise that products include fuzzy logic.

NOT AS FUZZY AS IT SOUNDS

Curiously, the results produced by fuzzy-logic systems are as precise and repeatable as those produced by respected traditional methods. Instead of indicating lack of precision, the term "fuzzy" more accurately refers to the way real-world sets have gradual boundaries.

When we say "the temperature is warm," there is not a specific temperature at which this expression goes from completely false to completely true. Instead, there is a gradual or fuzzy boundary, which requires a non-binary description of truth. In fact, the fuzzy logic definition for a set contains more information than the conventional binary definition of a set.

In conventional systems, the range of an input parameter is broken into sets that begin and end at specific values. For example, a temperature range described as warm might include the temperatures 56–84°F (see Figure 1a). The trouble with this thinking is that the temperature 84.01°F suddenly stops being considered warm. This abrupt change is not the way humans think of concepts like "temperature is warm."

Fuzzy logic uses a two-dimensional membership function to express the meaning of an input parameter such as "temperature is warm." Figure 1b shows how to express the meaning of warm temperature in a fuzzy-logic system. The x-axis shows the range of possible values for the input parameter temperature. The y-axis shows the degree to which temperature can be said to be warm (the degree of truth for the expression "temperature is warm"). The y-axis ranges from \$00 (not at all true) to SFF (completely true). You may see references where the degree of truth varies between 0.00 and 1.00, but in an embedded microcontroller, it is more practical to treat the truth value as an 8-bit binary value between \$00 and SFF.

OVERALL STRUCTURE OF A FUZZY KERNEL

Figure 2 shows a block diagram of a fuzzy-logic inference program in an embedded controller. Preprocessed system inputs enter the top of the fuzzy-inference kernel and system outputs leave at the bottom. The three processing blocks in the fuzzy kernel are executed in series each time the fuzzy kernel is called.

For each of the three processing blocks in the fuzzy kernel, there is a corresponding data structure in the knowledge base. Fuzzification compares the current value of system inputs against the input membership functions to determine values for fuzzy inputs stored in 8-bit RAM locations.

As rules from the rule list are processed, current fuzzy input values are used. The resulting values are stored in the fuzzy output locations in a second RAM array. Finally, the fuzzy output values are combined in the defuzzification step to produce system output values.

The fuzzy-inference kernel and the knowledge base can be developed independently. The advantage to this is that the microcontroller programmer developing the fuzzy kernel doesn't need to be familiar with the process to be controlled. Similarly, the process expert doesn't need to be a microcontroller programmer.

All that is necessary is that they agree on some basic ground rules such as number of inputs and outputs, number of labels for each input and output, and some basic limitations on rule structure. The fuzzy kernel can even be developed by a third party such as a semiconductor manufacturer or a fuzzy-development-tool vendor. The kernel software described in this article is an example of a fuzzy kernel developed without

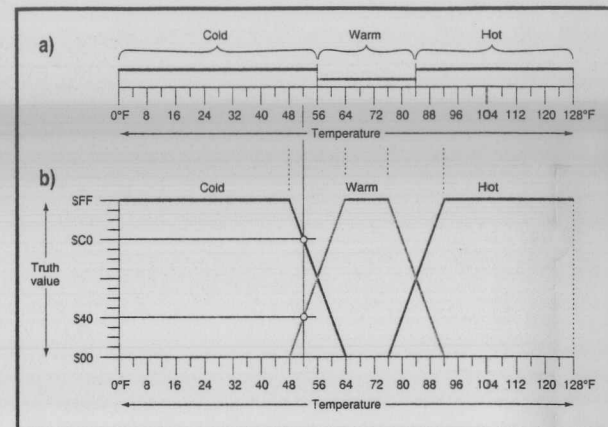


Figure 1—Traditional sets are simply defined by their endpoints. Fuzzy sets add a second dimension to express the degree of truth (on the y-axis), which allows sets to be defined with gradual boundaries between false and true.

any detailed knowledge of the systems in which it will be used.

EXPRESSING EXPERT KNOWLEDGE

For years, researchers have struggled to translate human knowledge into a form which can be manipulated by computers. If researchers could express the meaning of an idea like "temperature is warm" in an unambiguous numerical way, a digital computer could use this knowledge to make decisions similar to those made

by competent humans. Fuzzy logic has taken a giant step in this direction with the introduction of membership functions.

A fuzzy logic system is programmed with a series of rules such as "If temperature is warm and pressure is medium, then heater is full on." This natural-language control rule is simple enough for a human expert. Although conventional digital systems have trouble dealing with concepts like "temperature is warm," fuzzification gets around this problem by

assigning a concrete number between \$00 (false) and SFF (true) to this linguistic expression so that the microcontroller can process it further.

FUZZIFICATION

In this step, the current value of each input is compared to the membership functions for each label of the corresponding input. From this, it is possible to determine a numerical truth value for every label of every input. Input signals are typically preprocessed sensor signals scaled to

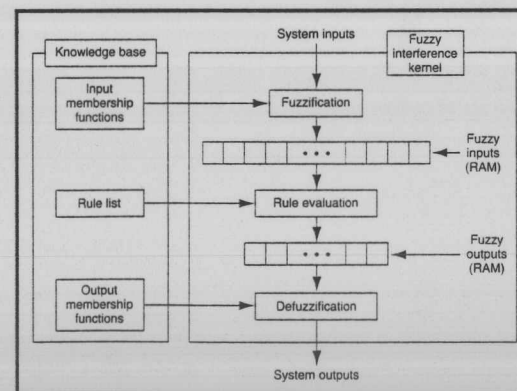


Figure 2—The knowledge base in this block diagram is developed by an application expert and the inference kernel, by an MCU programmer. Note the relationships between data structures in the knowledge base and the three main processes in the kernel. Fuzzy-input and fuzzy-output RAM data structures hold intermediate results during execution.

fit in the range from \$00 to \$FF. Pre-processing sensor inputs is an ordinary part of any embedded-control application, and fuzzy logic does not require any special skills for this job.

The inputs to the fuzzification process are the current 8-bit value of each system input and a membership function definition for each linguistic label of each system input. Results of the fuzzification step are fuzzy inputs in RAM—there's one byte for each label of each system input.

The fuzzy kernel in this article uses trapezoidal membership functions defined by two points and two slopes per membership function in nonvolatile memory. In other words, in an application with two system inputs and five labels per input, there would be 10 membership functions (4 bytes each = 40 bytes of ROM or EEPROM) and 10 fuzzy inputs (1 byte each = 10 bytes of RAM).

The major processing element in this step is a routine to determine the y-intercept on the membership function for one label corresponding to the

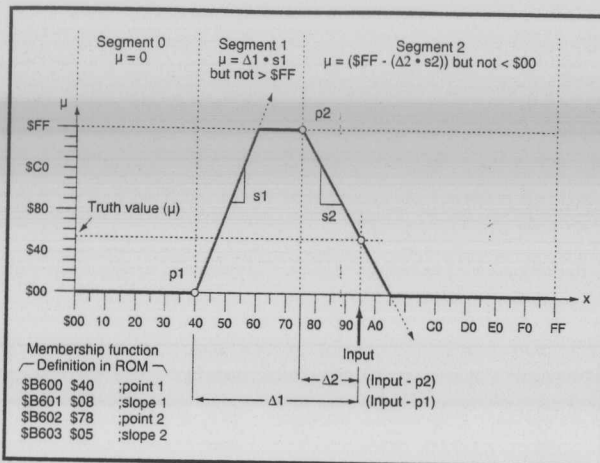


Figure 3—This figure shows one way of defining and evaluating a trapezoidal membership function. Segment 0 is defined by the position of point 1, segment 1 is defined by the values of point 1 and slope 1, and segment 2 is defined by the values of point 2 and slope 2.

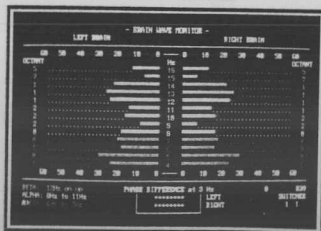
current value of one system input. Place this routine inside of two concentric loops. The inner loop executes once for each label of one input.

The outer loop executes once for each system input. In a system that has two inputs with five labels each, the outer loop executes twice and the

HAL-4

EEG Biofeedback Brainwave Analyzer

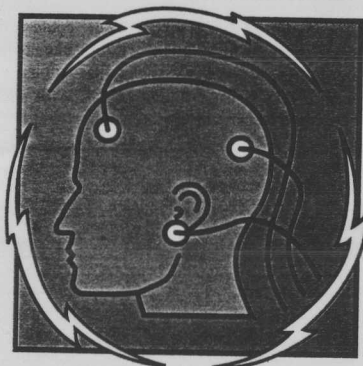
The HAL-4 kit is a complete battery-operated 4-channel electroencephalograph (EEG) which measures a mere 6" x 7". HAL is sensitive enough to even distinguish different conscious states—between concentrated mental activity and pleasant daydreaming. HAL gathers all relevant alpha, beta, and theta brainwave signals within the range of 4–20 Hz and presents it in a serial digitized format that can be easily recorded or analyzed. HAL's operation is straightforward. It samples four channels of analog brainwave data 64 times per second and transmits this digitized data serially to a PC at 4800 bps. There, using a Fast Fourier Transform to determine frequency, amplitude, and phase components, the results are graphically displayed in real time for each side of the brain.



HAL-4 KIT.....NEW PACKAGE PRICE - \$279 +SHIPPING
Contains HAL-4 PCB and all circuit components, source code on PC diskette, serial connection cable, and four extra sets of disposable electrodes.

to order the HAL-4 Kit or to receive a catalog,
CALL: (203) 875-2751 OR FAX: (203) 875-2204
CIRCUIT CELLAR KITS • 4 PARK STREET
SUITE 12 • VERNON • CT 06066

* The Circuit Cellar Hemispheric Activation Level detector is presented as an engineering example of the design techniques used in acquiring brainwave signals. This Hemispheric Activation Level detector is not a medically approved device, no medical claims are made for this device, and it should not be used for medical diagnostic purposes. Furthermore, safe use requires HAL be battery operated only!



New! ROM-DOS 6

A fully compatible, ROMable DOS for embedded systems

Gives you MORE... and LESS...

It Runs Windows!

► **More Embedded Tools:** ROMable EXE's, Automated BUILD Configuration, royalty free miniBIOS, optional Stackerc data compression and PCMCIA support.

◀ **Less ROM and RAM Used:** ROM-DOS is half the size of MS-DOS®.

► **More Flexibility:** Configure ROM-DOS the way you want. Device drivers in source let you support non-standard features or calls.

◀ **Less Money per Copy:** Save up to 80% compared to MS-DOS.

► **More Support:** Our technical staff assists you until your system is up and running, even on non-DOS issues.

◀ **Less Risk:** No-nonsense 90 day, 100% Money Back Guarantee on ROM-DOS Software Developer's Kit.

STACKER
VS
COMPRESSION

yes
It runs with
NetWare

FREE BOOTABLE DEMO DISK! CALL 1-800-221-6630



CARDTRICK™

PCMCIA MEMORY CARD FILE MANAGER

The only Flash File System to support both PCMCIA standards!

PCMCIA has approved two standards for Flash File systems. Why choose compatibility with only half the market? With CardTrick you can run them both simultaneously! Original Equipment Manufacturers use the CardTrick Software Developer's Kit to easily install the PCMCIA Memory Card File Manager on any x86 compatible hardware.

- ✓ Complies with new PCMCIA Flash standards
- ✓ FTL & FFS2 Flash Memory File Systems
- ✓ Works with or without Card Services and/or Socket Services
- ✓ One drive letter per slot for all cards
- ✓ Load as a Device Driver, TSR or BIOS Extension
- ✓ Low Cost Royalties
- ✓ Boot DOS from Flash

90-DAY MONEY BACK GUARANTEE ■ CALL 1-800-221-3630

E-MAIL: SALES@DALIGHT.COM

307 N. OLYMPIC, SUITE 201 • ARLINGTON, WA 98223 USA • (360) 435-8086 • FAX (360) 435-0253

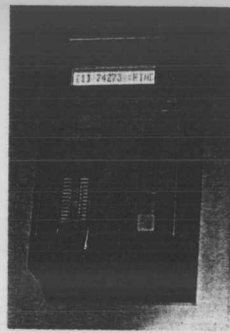
Datalight®

EMBEDDED AND MOBILE x86 SYSTEMS SOFTWARE

MS-DOS and Microsoft are registered trademarks of Microsoft Corporation. Developer tested only. Novell makes no warranties with respect to this product. The PCMCIA logo is a registered trademark of the Personal Computer Memory Card International Association. Stackerc is a registered trademark and LBS is a trademark of Sbc Electronics. ROM-DOS is a trademark of Datalight.

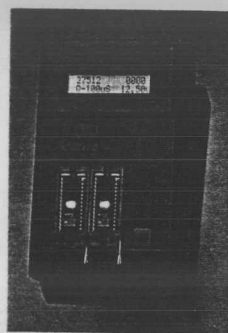
DATA Genie™

Data Genie offers a full line of test & measurement equipment that's innovative, reliable and very affordable. The "Express Series" of stand-alone, non-PC based testers are the ultimate in portability when running from either battery or AC power. Data Genie products will be setting the standards for quality on the bench or in the field for years to come.



HT-28 Express DIGITAL IC TESTER

The HT-28 is a very convenient way of testing Logic IC's and DRAM's. Tests most TTL 74, CMOS 40/45 and DRAM's 4164-414000, 44164-441000. It can also identify unknown IC numbers on TTL 74 and CMOS 40/45 series with the "Auto-Search" feature.
\$189.95



HT-14 Express EPROM PROGRAMMER

The HT-14 is one-to-one EPROM writer with a super fast programming speed that supports devices from 2732B to 27080, with eight selectable programming algorithms and six programming power (VPP) selections.
\$289.95



P-300 PC INTERFACE CARD PROTECTOR

The Data Genie P-300 is a useful device that allows you to quickly install add-on cards or to test prototype circuits for your PC externally. Without having to turn off your computer to install an add-on card, the P-300 maintains complete protection for your motherboard via the built-in current limit fuses.
\$349.95

MING
Microsystems
Division of MING & P, INC.
17921 Rowland Street
City of Industry, CA 91748
TEL: (818) 912-7756
FAX: (818) 912-9598

Call for a dealer near you.
1-800-473-6606

Data Genie products are backed by a full 1 year limited factory warranty.

©1994 MING Microsystems. All rights reserved. Data Genie logo is a registered trademark of MING Microsystems.

CCDG01

routine inside the inner loop a total of 10 times (five times for each pass through the outer loop). Figure 3 shows the routine for finding the y-intercept for one label of an input.

Listing 1 shows a practical algorithm for finding the grade of membership for one label of one system input. This routine is embedded inside the inner loop of the fuzzification process and follows the pattern mentioned above. The outer loop executes twice while the inner loop circles 10 times.

This routine also updates two pointers. The first one (X) points at the 4-byte membership function definition in the knowledge base. The second one (Y) points at the RAM location where the fuzzy input (result) will be stored.

The calculations associated with segment 2 take slightly longer than those for segment 1, so the routine checks to see if the input is there first. This check helps balance the execution time and keeps the worst-case path as short as possible. The shortest path occurs when the input is in segment 0. Two range-checking sequences, BLS NOT_SEG2 and BLO HAV_GRAD, are in this path.

This algorithm uses the x-positions of two points and two unsigned slope values to define a membership function. While it would be more straightforward to describe a trapezoid with four corner points, it then requires at least one divide during run time. The method described here never needs to execute anything more difficult than one 8-bit multiply to find a grade of membership.

Trapezoidal membership functions are commonly used because they meet the requirement of providing a gradual transition from false to true while requiring only simple calculations to compute an intercept. Some programs only allow triangular membership functions, but trapezoids are just as easy to process. A trapezoid with a top width of zero makes a triangular membership function.

RULE EVALUATION

Although rules sound like arbitrary, natural-language statements, they follow a fairly strict syntax. The typical fuzzy-logic kernel in a small,

Listing 1—This routine performs fuzzification for one label of one system input. Refer to Figure 3 while studying this program.

```
*GET_GRADE—Routine to project a current input value onto
* an associated input membership function (fuzzification).
* Result is stored to fuzzy input and pointers are updated.
* ENTRY VALUES: A = Current system input value
* X = Pointer to membership function in ROM
* Y = Pointer to fuzzy input in RAM
* EXIT VALUES: A = unchanged (ready for next GET_GRADE call)
* B = used internally to calculate grade (result)
* X = add 4 to point at next MF definition
* Y = add 1 to point at next fuzzy input in RAM
```

```
GET_GRADE PSHA      :Save input value of A
          CLR8       :In case grade = 0
          SUBA 2,X    :Input value - pt2 -> A
          BLS NOT_SEG2 :If input < pt2
          LDAB 3,X     :Slope 2
          BEQ HAV_GRAD :Skip if zero slope
          MUL HAV_GRAD :((In - pt1) * slp2 -> A:B
          TSTA        :Check for > $FF
          BEQ NO_FIX   :If upper 8 = 0
          CLR8        :Limit grade to 0
          BRA HAV_GRAD :In limit region of seg 2
NO_FIX    SUBB $FF     :B - $FF
          NEGB        :$FF - B
          BRA HAV_GRAD :($FF - ((In - pt2) * slp2))
NOT_SEG2  ADDA 2,X     :Restore input value
          SUBA 0,X     :Input value - pt1 -> A
          BLO HAV_GRAD :In < pt1 so grade = 0
          LDAB 1,X     :Slope 1
          BEQ ZERO_SLP :Skip if zero slope
          MUL          :((In - pt1) * slp1 -> A:B
          TSTA        :Check for > $FF
          BEQ HAV_GRAD :Result OK in B
ZERO_SLP  LDAB $FF     :Limit region or zero slope
HAV_GRAD  INX         :Point at next MF spec
          INX
          INX
          INX
          STAB 0,Y     :Save one fuzzy input
          INY         :Point at next fuz input
          PULA        :Restore A register
```

embedded-control system limits rules to the following form:

```
IF system_input_x is label_a
AND system_input_y is label_b
THEN output_w is label_c.
```

Each of the linguistic expressions like "system_input_x is label_a" corresponds to a specific fuzzy input value in RAM. These values are determined by the fuzzification step. The expression "system_output_w is label_c" corresponds to a specific fuzzy output. AND is a fuzzy operator which corresponds to the mathematical minimum operation. All the linguistic expressions on the left side of the rule are connected by ANDs. The truth value

for the whole rule is the value of the smallest fuzzy input on the left side. There is an implied OR between successive rules, which corresponds to the mathematical maximum operation.

Before processing the rules, all fuzzy outputs are initialized to \$00 (meaning not true at all). As rules process, the truth value for the current rule is stored in each fuzzy output on the right side of the rule unless the fuzzy output is already bigger (this is the maximum operation).

Rules can be stored in the knowledge base as a simple list of pointers to fuzzy inputs and fuzzy outputs. For the kernel described in this article, a 7-bit offset from the start of the fuzzy input array is used for each rule antecedent.

Tight Budget '51 Emulation

8051 Family Emulator is truly Low Cost!

The DryICE Plus is a modular emulator designed to get maximum flexibility and functionality for your hard earned dollar. The common base unit supports numerous 8051 family processor pods that are low in price. Features include: Execute to breakpoint, Line-by-Line Assembler, Disassembler, SFR access, Fill, Set and Dump Internal or External RAM and Code, Dump Registers, and more. The DryICE Plus base unit is priced at a meager **\$299**, and most pods run only an additional **\$149**. Pods are available to support the 8031/2, 8751/2, 80C154, 80C451, 80C535, 80C537, 80C550, 80C552/62, 80C652, 80C851, 80C320 and more. Interface through your serial port and a comm program. Call for a brochure or use INTERNET. We're at info@hte.com or [ftp at ftp.hte.com](http://ftp.hte.com)

Tighter budget? How about \$149 emulation?

Our **\$149** DryICE model is what you're looking for. **Not** an evaluation board - much more powerful. Same features as the DryICEPlus, but limited to just the 8031/32 processor.

You can afford it!

So, if you're still doing the **UV Waltz** (Burn-2-3, Erase-2-3), or debugging through the limited window ROM emulators give, **call us now** for relief! Our customers say our products are **still** the **best** Performance/Price emulators available!

Look into our Single Board
Computer solutions, too!

HTE HiTech Equipment Corp.
9400 Activity Road
San Diego, CA 92126
(Fax: (619) 536-1458)

Since 1983

(619) 566-1892



Internet e-mail: info@hte.com
Internet ftp: ftp.hte.com

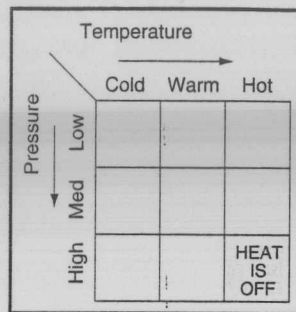


Figure 4—All possible combinations of input conditions are summarized in this course rule matrix. The shaded cell represents the rule "If temperature is hot and pressure is high, then heat is off."

The MSB of all antecedent pointers is clear. A byte with the MSB set plus a 7-bit offset from the start of the output array is used for each rule consequent.

Since the MSB distinguishes consequents from antecedents, rules may have any number of inputs or outputs. It would be faster, but less flexible, to define rules with a fixed structure such as two antecedents and one consequent. It would also be faster to use

whole addresses rather than offsets in the rule list, but that would more than double the amount of memory required for the rule list.

Since each input has only a finite number of labels, there are only a certain number of possibilities for unique rules. A system with two inputs, each having three labels, has a maximum of nine possible rules as shown in Figure 4. As you can see, the treatment of values is very coarse. No transition regions are shown between adjacent labels of the inputs.

Figure 5 corrects this. It shows the membership functions below and to the right of the rule matrix. This figure shows the areas where more than one label of an input is true at the same time. The knowledge base only specifies the system-output level at the nine shaded cells of Figure 5. The other cells represent combinations of input values that cause two or four rules to be true to some degree at the same time. In these areas, the defuzzification step combines the recommended actions of all of the contributing rules.

DEFUZZIFICATION

After the rule-evaluation step, each of the fuzzy outputs has a value corresponding to the degree that output action should be applied. These can be considered as recommendations for the system-output level. The defuzzification step combines these separate recommendations into a single, composite system-output value.

The program in this article uses singleton membership functions,

which are simply the x-axis position of one label of a system output. The fuzzy output value in RAM represents the height (y value) of this membership function or the degree to which it should apply. The following formula shows the calculation needed for defuzzification:

$$\frac{\sum_{i=1}^n F_i \times S_i}{\sum_{i=1}^n F_i}$$

where n is the number of fuzzy outputs associated with system output, F_i is a weight (fuzzy output value from runtime RAM), and S_i is a membership-function singleton position (from the knowledge base). The result of this calculation is the system-output action. F_i and S_i are 8-bit values and the value of n is typically 8 or less. This makes the numerator a 19-bit value and the denominator an 11-bit value.

Normally, a 19-bit by 11-bit divide yields up to a 19-bit result. But in our case, the values are not independent and we know the result fits in an 8-bit number. Figure 6 shows the defuzzification process graphically.

AN ALTERNATE OFF-LINE APPROACH TO FUZZY LOGIC

When fuzzy logic was first introduced, it was thought to require a lot of processing horsepower. If you choose to use floating-point calculations and complex shapes for membership functions, this is true.

By using simple shapes such as trapezoids and singletons, we greatly simplify the calculations for fuzzification and defuzzification. By using fixed-point calculations in which truth varies between

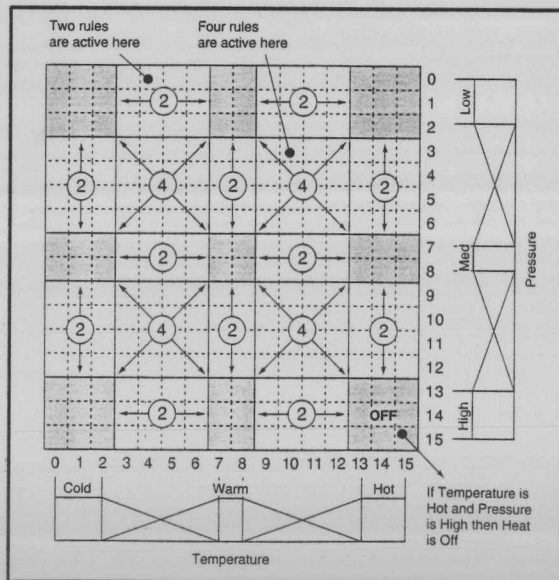


Figure 5—A more detailed view of the rule space shows the areas where more than one rule can be active at a time. Membership functions for temperature and pressure are shown below and to the right of the rule space.

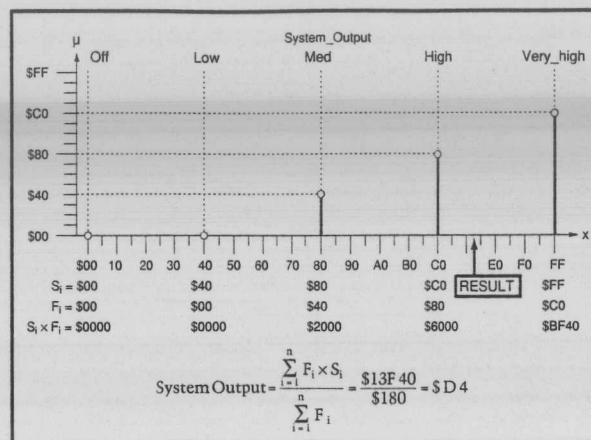


Figure 6—This figure demonstrates the defuzzification process in which three fuzzy outputs are active at the same time to different degrees. The result is the weighted average of all active fuzzy outputs.

\$00 and \$FF, we eliminate the need for floating point.

Some of the first embedded-control applications for fuzzy logic used the more complex floating-point calculations running on a larger computer or workstation. An output value was calculated for every combination of inputs to derive a control surface. This control surface was then stored in the embedded controller as a large table. During operation of the application, current input values were used to look up the required output in the table.

Although this approach was fast, it tended to require a large memory for the control surface look-up table. It was also difficult to modify this type of system because you had to return to the workstation to make changes and generate a new control-surface table.

CONCLUSION

Fuzzy logic is a powerful and accessible tool for embedded-control applications. It offers a way to work with complex human concepts within a relatively small microcontroller program. This in turn makes it possible to solve problems previously thought to be too difficult for a small microcontroller.

Not surprisingly, many of the first fuzzy-logic applications are traditional control problems in which fuzzy logic

replaces another methodology such as PID. The more interesting applications involve new problems in which an embedded controller was previously unable to solve the problem using traditional digital techniques. ■

Jim Sibigroth is a system design engineer working on advanced microcontrollers for Motorola. Prior to his work on fuzzy logic, Jim was the systems project leader for the MC68HC11 and wrote the M68HC11 Reference Manual. Jim's other book, *Understanding Small Microcontrollers* (ISBN 0-13-089129-0), introduces working engineers to microcontrollers and assembly language programming. He may be reached at jims@seasick.sps.mot.com.

SOFTWARE

Software for this article is available from the Circuit Cellar BBS and on Software On Disk for this issue. Please see the end of "ConnecTime" in this issue for downloading and ordering information.

I R S

404 Very Useful
405 Moderately Useful
406 Not Useful

If you need I/O...

Add these numbers up:

- 80C552 a '51 Compatible Micro
- 40 Bits of Digital I/O
- 8 Channels of 10 Bit A/D
- 3 Serial Ports (RS-232 or 422/485)
- 2 Pulse Width Modulation Outputs
- 6 Capture/Compare Inputs
- 1 Real Time Clock
- 64K bytes Static RAM
- 1+ UVPRAM Socket
- 512 bytes of Serial EEPROM
- 1 Watchdog
- 1 Power Fail Interrupt
- 1 On-Board Power Regulation

It adds up to real I/O power!

That's our popular 552SBC, priced at just \$299 in single quantities. Not enough I/O? There is an expansion bus, too! Too much I/O? We'll create a version just for your needs, and pass the savings on to you! Development is easy, using our Development Kit: The 552SBC-50 Development board with ROM Monitor, and an 8051 C compiler for just \$449.

New Versions of the 8031SBC

Our popular 8031SBC can now be shipped with your favorite 8051 family processor. Models include 80C51FA, DS80C320, 80C550, 80C652, 80C154, 80C851 and more. Call for pricing today!

Truly Low-cost In-Circuit Emulator

The DryICE Plus is a low-cost alternative to conventional ICE products. Load, single step, interrogate, disasm, execute to breakpoint. Only \$448 with a pod. For the 8051 family, including Philips and Siemens derivatives. Call for brochure!

Call for your custom product needs. Quick Response.

HTE HiTech Equipment Corp.
9400 Activity Road
San Diego, CA 92126
(Fax: (619) 530-1458)
Since 1983
(619) 566-1892
VISA MasterCard
Internet e-mail: info@hte.com
Internet ftp: ftp.hte.com